Project Context for AI Agents

This file contains critical rules and patterns that AI agents must follow when implementing code in this project. Focus on unobvious details that agents might otherwise miss.

Technology Stack & Versions

Monorepo

- pnpm workspaces; shared `/apps` (web/api/marketing) + `/packages/theme`

Frontend (Client-only SPA)

- React `18.x` with Vite `5.x` (CSR SPA)
- TypeScript in scope
- React Router `6.x` (explicit routing; no file-based routing)
- MUI + Emotion
- Storybook (component workshop)
- **CRA is deprecated and not allowed**
- **Lockfile required (npm/pnpm/yarn) to prevent version drift**
- **Decision:** CSR SPA (no SSR) keeps hosting simple on SWA and matches MVP scope

Backend (API)

- Azure Functions runtime `v4`
- .NET isolated worker
- Default: Windows Consumption (MVP)
- **.NET SDK: `10.0.x` (LTS)**
- **Plan rule: if plan changes away from Windows Consumption, revalidate .NET support matrix**
- **Decision:** Windows Consumption keeps cost low and avoids Linux Consumption .NET constraints

Marketing

- Eleventy (11ty)

Testing

- Playwright E2E in `/tests/e2e`

Auth

- Azure Static Web Apps (SWA) built-in auth (EasyAuth) with Microsoft Entra ID
- **Do not introduce custom auth flows in MVP**
- **Decision:** EasyAuth reduces surface area and aligns with trust-first constraints

Data

- Azure Table Storage via `Azure.Data.Tables 12.x`
- Partitioning convention (call out early for MVP query efficiency):

    - `ProgrammeId` as `PartitionKey` for programme-scoped entities (Memberships, Cycles, Matches)
    - `RowKey` as deterministic ID (`UserId`, `CycleId`, `MatchId` per entity)
- **Avoid cross-partition scans in MVP endpoints**
- **Decision:** Programme-level queries are the dominant MVP access pattern

Notifications

- Azure Communication Services Email only: `Azure.Communication.Email 1.x`
- Teams notifications are separate (Graph or webhook), but Email is the guaranteed fallback
- **If Teams delivery fails, email remains the authoritative channel**
- **Decision:** Email is the reliable baseline; Teams is additive

Deployment

- GitHub Actions using SWA's standard CI/CD flow
- **Decision:** Use SWA's default workflow for lowest friction

Cost constraints (All-Free MVP)

- Two SWAs on Free tier
- No Key Vault in MVP (env/app settings + GitHub Secrets)
- Minimal logging (no App Insights unless needed)

Versioning rule

- "Latest" means **pinned** major streams (e.g., `18.x`, `5.x`) with lockfiles; avoid floating versions

Critical Implementation Rules

Language-Specific Rules

TypeScript (Frontend)

- `"strict": true`
- `"noUncheckedIndexedAccess": true`
- No `any` unless explicitly commented with intent and scope.
- Prefer discriminated unions over loose objects.
- Runtime validation for all external inputs (API requests, auth claims, query params). Types are not trusted at the boundary.
- **Auth claims are treated as untrusted input; validate presence and shape before use.**
- **Use a schema validator at boundaries (e.g., Zod) or a documented equivalent.**
- **If a schema is shared between FE/BE, keep it in a single source of truth to avoid drift.**
- **Risk guard:** `any` creeping in via third-party types requires an explicit comment.
- **Risk guard:** always validate external inputs (auth claims, API responses) with Zod.

C# (.NET Functions, Backend)

- Nullable reference types **enabled**.
- Treat warnings as errors for MVP.
- Async-only APIs (no sync-over-async).
- Explicit DTOs at boundaries (no leaking domain models over HTTP).

- Runtime validation for all inbound payloads (model binding is not sufficient on its own).
- **Fail fast with `400` on validation errors; no partial processing.**
- **Auth claims are treated as untrusted input; validate presence and shape before use.**
- **Nullability config must be consistent across projects to avoid warning floods.**
- **Mapping helpers are encouraged to keep DTO ↔ domain translations consistent.**
- **Risk guard:** always validate inbound DTOs; do not rely on model binding alone.

Testing and contracts

- Any new API shape requires at least one contract-level test.
- Tests validate runtime behaviour, not just type alignment.
- Focus on "does this break consumers?" rather than exhaustive coverage.
- **Contract tests include auth/claims edge cases and null/empty input.**
- **Contract tests prioritise boundary failure modes over happy-path breadth.**
- **When schemas change, include a test covering backward compatibility or explicit breaking change.**

Constraint

- These rules apply **at system boundaries**, not internally everywhere.
- Inside the domain, favour clarity over ceremony.
- **If a rule adds ceremony without boundary risk reduction, don't apply it.**

Framework-Specific Rules

React (SPA)

- No state management library in MVP. Use React hooks, lifting state, and Context only where genuinely shared.
- Keep routing minimal. Only introduce routes that map to real user tasks (e.g., Home, Programme, Account).
- Auth UX is invisible — no custom login screens beyond Entra redirects. No "welcome wizard".
- First-run is minimal: join the programme, see what happens next. No heavy onboarding UI.
- Prefer boring components over clever abstractions. Consistency beats custom UI patterns.
- **Client routing does not imply access — every route still requires server-side authorisation.**
- **Decision:** Keep SPA minimal to avoid accidental productisation and preserve trust-first UX.
- **Don't poll `/.auth/me` excessively; treat auth state as stable per session.**

React + MUI

- MUI theme must be sourced from `/packages/theme/muiTheme.ts` (single source of truth).
- No ad-hoc theme overrides outside the shared theme package.

Storybook

- Stories live alongside components (same folder) when feasible.
- No API calls inside stories; keep them deterministic.

Eleventy (11ty)

- Use `/packages/theme/tokens.css` as the base style contract.

- No JS-heavy client logic; keep marketing pages static.

SWA (Free tier constraints)

- No Key Vault in MVP; config via app settings.
- Role allowlist read from Table Storage (source of truth).

Azure Static Web Apps + EasyAuth

- Use SWA built-in auth end-to-end. Do not bypass it with custom token handling.
- The API trusts identity only via platform-provided headers/claims from SWA. No client-supplied identity.
- Authorisation is server-side. The client never decides access.
- **Validate auth claims on every request; do not assume presence/shape.**
- **Decision:** EasyAuth reduces attack surface and avoids bespoke auth risk.

Azure Functions (v4, isolated)

- Keep Functions thin: route, auth check, validate, call service, return.
- Avoid heavy middleware pipelines and avoid building a "mini web framework".
- Routing conventions: `/api/programmes`, `/api/programmes/{programmeId}`, `/api/matches`, `/api/me`.
- **Routes are noun-based; avoid action verbs in paths.**
- Prefer minimal DI. Only inject what you need; keep services small and explicit.
- Idempotency for scheduled operations (matching runs) — reruns must not create duplicate matches.
- **Idempotency key = `ProgrammeId + CycleId` for matching runs.**
- **Authorisation checks must be explicit in every handler; no "global" implicit access.**
- **Decision:** Thin handlers and explicit auth keep operational risk low and debugging straightforward.
- **OpenAPI.NET is required for Functions OpenAPI generation; specs must stay in sync with DTOs and contract tests.**
- OpenAPI is generated, not hand-written.
- Use attribute-based annotations only where default inference is insufficient.
- One spec per Functions app; no versioning in MVP (single spec, single version).
- Operation IDs must match handler names (e.g., `CreateProgramme`, `JoinProgramme`).
- Schemas generated from DTOs only; no anonymous or inline schemas.
- Hide any internal or admin-only endpoints explicitly.

Azure Table Storage

- Access only through a thin repository layer that enforces PartitionKey rules.
- Always query by PartitionKey first. Design APIs so you never need full-table scans.
- Stable table names. No per-programme tables in MVP.
- Partition strategy is a rule, not a suggestion:
  - Programme-scoped entities partitioned by `ProgrammeId`.
  - User-scoped entities partitioned by `UserId` only if you truly need user-first queries.
- Avoid cross-partition joins. If you need them, your model is wrong for Table Storage.
- **Enforce "programme boundary" checks in repository methods to prevent cross-programme data leakage.**

- **Decision:** Programme-scoped partitions match dominant query patterns and reduce accidental data leakage.

UX guardrails

- Reduce perceived effort: default views show “what to do next” and nothing else.
- Avoid anything that feels like monitoring: no activity feeds, no “stats” screens in MVP.
- **Decision:** MVP success depends on trust and low perceived burden.

Testing Rules

Testing priorities (in order)

- Unit tests first.
- Contract tests next.
- Integration tests only where they buy confidence cheaply.
- E2E tests only where absolutely necessary - avoid flake by default.
- **Decision:** MVP risk is at boundaries and logic, not UI polish.

Boundary-first rule

- Every endpoint must have tests for:
  - Missing/invalid identity or claims.
  - Invalid payloads (shape, required fields, unexpected values).
  - Authorisation outcomes (403/401/404 as appropriate).
- **Contract tests must assert claim-based authorisation decisions, not just HTTP status codes.**
- **Include negative tests for privilege escalation (e.g., cross-programme access).**
- **Decision:** Trust failures are costliest; test them first.

Contract discipline

- Any API shape change must update or add at least one contract test.
- Tests validate runtime behaviour, not just TypeScript type alignment.
- **Decision:** Contracts are the change-control surface for a small team.
- OpenAPI specs must align with contract tests.
- Any API shape change requires: updated DTO, updated OpenAPI output, updated contract test.
- **Contract tests must assert ISO 8601 date formats and Problem Details error shape.**

Table Storage tests

- Keep Table Storage behind a repository.
- Add a thin integration test using Azurite (or equivalent) only if it stays simple.
- No complex test harnesses for MVP.
- **Repository tests must include cross-partition access denial.**
- **Decision:** Integration only where it confirms partition rules cheaply.

Coverage stance

- Do not chase 100% coverage.
- Prioritise: matching correctness, idempotency, boundary auth, and top failure paths.
- **Idempotency tests must cover duplicate match creation and duplicate notifications.**

- **Decision:** Coverage serves trust and reliability, not vanity metrics.

Trust-promise guard

- Guard suite asserts we aren't drifting into surveillance:
  - No endpoints returning individual-level analytics.
  - Programme-level views return aggregated, non-identifying data only.
  - Any attempt to access user-level participation metrics is rejected (403/404).
- Treat failures in this suite as build blockers.
- **Guard suite must scan new endpoints so coverage doesn't regress.**
- **Guard suite must include a "cross-programme data leak" check.**
- **Decision:** Anti-surveillance is a core product promise.

Flake policy

- If a test is flaky twice, fix or delete it. No "ignore and move on".
- **Decision:** Flaky tests erode trust faster than no tests.

Code Quality & Style Rules

Core principle

- Consistency over cleverness. Prefer obvious code a new dev can follow in one pass.

Abstractions

- No abstraction until the second real use.
- **When introducing an abstraction, document the two concrete usages it replaces.**
- No "frameworks inside the app" for MVP (custom pipelines, generic repositories, meta-config engines).
- **If an abstraction adds indirection without reuse, remove it.**

API contracts

- Contracts are explicit and stable.
- DTOs at the boundary. No leaking domain models over HTTP.
- Avoid internal jargon in public interfaces unless deliberate and documented.
- Versioning is avoided in MVP by keeping scope small, but breaking changes are treated as serious.

Naming conventions

- API handlers: `VerbNoun` aligned to route and intent (e.g., `CreateProgramme`, `JoinProgramme`, `RunMatchingCycle`).
- DTOs: `{Thing}{Verb}Request` / `{Thing}{Verb}Response` where useful.
- Table entities: `{EntityName}Entity` with clear keys (`PartitionKey`, `RowKey`) and consistent metadata fields.
- Tables: stable names, plural, consistent casing.
- No naming exceptions by default.
- Acronyms allowed only if organisation-wide (e.g., ID, URL).
- No legacy terms unless explicitly documented at the boundary where they appear.
- **If a term is ambiguous, prefer the longer, clearer name.**

Comments and documentation

- Comment the *why*, not the *what*.
- Comments required at boundaries where trust, security, or data-handling decisions are made.
- Keep comments short; if it needs a paragraph, write a small markdown doc instead.
- **Boundary docs should be colocated (README in folder or short `docs/` note).**
- **If a boundary decision changes, update the doc in the same PR.**
- **C# projects must enable XML documentation; public APIs must be documented.**
- XML docs required on:
  - Public DTOs.
  - Public API handlers.
  - Any method making a trust, security, or data-retention decision.
- Focus on intent and guarantees, not implementation detail.
- No documentation required for private helpers unless the logic is non-obvious.

Tooling

- Do not pin tooling now.
- Use sensible defaults when introduced.
- No custom rule sets or bikeshedding for MVP.
- If ESLint, Prettier, or EditorConfig are added, they must align with clarity, consistency, and low friction.

Code review guardrails

- If a change increases cognitive load without increasing user value, reject it.
- If a change introduces user-level tracking or "analytics temptation", reject it unless explicitly in scope (it isn't for MVP).
- **Prefer small, single-purpose diffs; avoid mixed refactors + feature changes in one PR.**
- **If a PR touches boundaries, require explicit reviewer attention to trust/privacy impact.**

Development Workflow Rules

Definition of Done (MVP) A change is not done unless:

- Tests updated or added where behaviour changed.
- Contract tests updated for any API shape change.
- OpenAPI output updated and aligned with DTOs.
- Trust-guard test suite passes.

Boundary change discipline

- Any change that touches an external boundary (API, auth, storage schema, notification payloads) must update:
  - Contract tests.
  - OpenAPI spec.
- If this feels heavy, the change is probably too large for MVP.

PR hygiene

- Every PR must include a short summary answering:

    - *What behaviour changed?*
    - *Who is affected?*
    - *Is this user-visible or internal only?*
- Any UX-affecting change must state the intended user experience in plain language.

Scope protection

- No tracking or analytics changes without explicit scope approval.
- No drive-by refactors in MVP.
- Refactors only when:
    - They unblock a required change, or
    - They remove proven complexity introduced earlier.

Velocity guardrail

- Prefer two small PRs over one "clever" one.
- If a PR can't be explained in a short paragraph, it's probably too big.

Commit granularity & history hygiene

- One agent change = one commit.
- PRs should tell a coherent story; avoid merging many micro-commits.
- **Squash on merge** to keep main clean (retain detailed commits in PR for audit).

Branching model (AI agents)

- Agents never commit directly to `main`.
- Use `ai/<topic>` branches per agent task.
- Scope branches tightly to avoid conflicts (one area per branch).

Guardrails on what agents can change

- Agents may change code and docs.
- Agents must NOT change security/auth flows, data model keys/partition strategy, logging/telemetry, or role/authorisation logic unless:
    - The PR explicitly declares it, and
    - Contract tests + OpenAPI + trust suite are updated/passing.

Deterministic formatting

- Add `.editorconfig` now.
- If Prettier/ESLint are added later, pin versions and run formatting in CI.

AI change metadata

- PR description includes prompt/goal + key decisions.
- Optional: maintain `docs/ai-changelog.md` for major AI-driven changes.

Revert strategy

- If an AI commit breaks something, revert immediately rather than fix-forward in the same PR.
- Keep rollback clean.

Boundary test gating

- For PRs touching API/auth/storage/notifications: update contract tests, update OpenAPI, run trust-guard suite, and run Playwright smoke (desktop + mobile match flow).

Critical Don't-Miss Rules

Data isolation

- No cross-programme data leakage under any circumstances.
- Every Table Storage query must enforce `ProgrammeId` as `PartitionKey` where applicable.
- Any query that could span programmes is a bug, not a feature.

Matching discipline

- No "smart", weighted, or AI-driven matching in MVP.
- Random or lightly constrained matching only.
- If matching logic needs explanation beyond a sentence, it's out of scope.

Anti-mandate principle

- Participation is never required.
- The programme must not be positioned as expected, encouraged by managers, or linked to performance.
- Opt-out is always easy and consequence-free.

Visibility limits

- No manager-level visibility into individual participation.
- No reports that could be used to infer individual behaviour.
- Sponsors see only aggregate, non-identifying signals.

Analytics-temptation guard

- Any endpoint or change that could enable surveillance or individual-level analytics is blocked by default.
- Such changes require explicit re-scoping (which MVP will not do).

Trust contract

- If any of these rules are broken, the product fails even if everything else works.

Usage Guidelines

For AI Agents:

- Read this file before implementing any code
- Follow ALL rules exactly as documented
- When in doubt, prefer the more restrictive option
- Update this file if new patterns emerge

For Humans:

- Keep this file lean and focused on agent needs
- Update when technology stack changes
- Review quarterly for outdated rules
- Remove rules that become obvious over time

Last Updated: 2026-01-14